Australian Government

Department of Defence
Defence Science and
Technology Organisation

Performance Analysis and Optimisation of the Fact Extractor System

Shona Heath

DSTO-TN-0566

Australian Government
Department of Defence
Defence Science and
Technology Organisation

Performance Analysis and Optimisation of the Fact Extractor System

Shona Heath

Command and Control Division
Information Sciences Laboratory

DSTO-TN-0566

ABSTRACT

DSTO has developed an in-house application called the Fact Extractor System for performing Information Extraction. This system can be used to extract interesting information from text documents. It is a component-based system providing a suite of tools to do this task. The system has a number of deployment tools, including one called FormFiller. FormFiller is an application that enables a user to process a set of documents, one at a time interactively or in an automated batch mode, with one or more Fact Extractors and save results to an output file under user control. This report describes performance testing and optimisation of the FormFiller application and Fact Extractors.

AQ F04-11-1382

Published by

DSTO Information Sciences Laboratory
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-013-124
June 2004

Performance Analysis and Optimisation of the Fact Extractor System

Executive Summary

The Intelligence Community often needs to process large volumes of unstructured text to obtain interesting information. Processing the text manually is not a very efficient use of time. The DSTO Fact Extraction System was developed to demonstrate how automated text processing could improve the efficiency of finding interesting information

Using regular expression technology, tools known as Fact Extractors have been developed. Fact Extractors contain rules that specify the type of fact to be discovered, and are then run over the text document to highlight the facts.

The FormFiller application allows users to run one or more Fact Extractors over a document or set of documents at once and save results to an output file. The application can be deployed automatically to capture and record all of the facts from the chosen Fact Extractors, or run under user control.

The performance of the FormFiller application was an issue as clients interested in the Fact Extractor suite of tools had complained that it was too slow. A series of tests were performed to record the performance of FormFiller. FormFiller was written in the Java programming language, and a Java performance analysis tool was used to identify areas of the source code that were taking up a lot of processing time. The code was then inspected and changed. The same performance tests were run over the new code to see if they had an effect on performance. The first had a significant improvement on processing time. This process was then repeated, and another area of the program was identified and improved.

The improvements made have benefited both FormFiller and the entire suite of Fact Extractor tools, as some of the problems were in code used by the entire system. The advantage is that potential clients and users of the system will no longer be discouraged by long document processing times.

The performance of different Fact Extractors in FormFiller was then measured, as this will assist in future Fact Extractor design.

This report describes the approach taken in analysing and tuning the performance of the Fact Extractor system, and is aimed at readers with a technical interest in the system.

Author

Shona Heath
Command and Control Division

IBL Student, Swinburne University of Technology
Studying Bachelor of Engineering (Telecommunications and Internet Technologies) / Bachelor of Applied Science (Computer Science and Software Engineering)

Contents

1. Introduction

The DSTO Fact Extractor System is an application developed to perform information extraction over unformatted text [1]. It comprises a set of components to assist with this task. The FormFiller application is part of the Fact Extractor System tool suite [2]. It is an end user tool for deploying one or more Fact Extractors over a set of documents to extract interesting information.

One of the most common user complaints about the DSTO Fact Extractor system, in particular the FormFiller application, was that the performance was very slow, especially over large groups of documents.

A series of tests were performed to identify potential causes for slow performance of the system. A standard set of test documents was developed initially and all tests were performed over these documents.

As the Fact Extractor system is highly reliant on the Java Regular Expression (REGEX) package, the performance of FormFiller can be no faster than the package's regular expression pattern matcher. A test was performed to determine this theoretical lower bound on performance.

Secondly, a baseline was produced, which involved tests to determine the current performance of FormFiller. This baseline was later used to check if any changes to the system had resulted in an increase in performance or not.

Once the baseline was established, a Java performance analysis tool was used to target hot spots where a lot of processing time was being spent. Changes were then made to the source code, and the performance test was repeated, until members of the Fact Extractor team were happy with the results. The goal was for FormFiller to be able to process 1000 small (i.e. less than 10 KB each in size) documents within a minute using the computer configuration described in Section 2.4.

Once the goal was achieved, the performance of a range of different fact extractors was measured, as the results will assist in designing fact extractors that are efficient to process.

Execution time was the main criteria in these tests, however memory usage was also observed.

This report details the approach outlined above along with the outcomes of the performance optimisation. It is directed at readers with a technical interest in the Fact Extractor System.

2. Tests Performed

2.1 Test Corpora

To facilitate testing, two representative sets of documents were derived from the PROMED[1] corpus. They were designed to test FormFiller's performance in handling many small documents as well as large documents.

- **1000smalldocs** – This folder contains 1000 small text files with sizes ranging from 1 KB to 7 KB.
- **10largedocs** – This folder contains 10 large text files with sizes ranging from 64 KB – 100 KB.

For tests that used HTML input, the PROMED data was available in HTML format, and two more corpora, **1000smallHTML** and **10largeHTML**, were derived.

2.2 Estimation of Theoretical Lower Bound

Fact Extractors use regular expressions to find matches in text. One regular expression is evaluated for every sentence per rule. It was decided to run the regular expression engine by itself, as this gave an indication of the theoretical best performance. The performance of FormFiller cannot be any faster than the performance of the regular expression pattern matcher it uses. In this case, the theoretical absolute lowest bound for time would be the performance of the Java REGEX pattern matcher. The first test determined the performance of this pattern matcher. This gave an indication of how much extra overhead and processing time the FormFiller added.

To first test the Java REGEX package, a program called 'TesterGUI' that was previously developed by a former Swinburne University of Technology IBL[2] student was used. This program was originally designed to compare the Java REGEX package with other regular expression pattern matchers, and for the purposes of this test it could be reused and configured to match using Java REGEX.

The following regular expression was used for this test. This regular expression was contained in a text file used with TesterGUI, but because FormFiller uses fact extractors, this regular expression was made into a fact extractor called 'testdate.fx'. The regular expression was:

[1] PROMED is a collection of medical publications containing information about different diseases.

[2] IBL – Industry Based Learning, a program run through Swinburne University of Technology, http://www.swin.edu.au

```
((?:Mon|Tue|Wed|Thu|Fri|Sat|Sun)(?:[a-z]+)?),?([0-3]?[0-9])
(Jan|Feb|Mar|Apr|May|Ju|Aug|Sep|Oct|Nov|Dec)(?:[a-z]+])?  ([0-
9]+)
```

In both FormFiller and TesterGUI, the test was run over the '10largedocs' corpus.

The 'TesterGUI' application automatically tests each document in the corpus against the regular expression pattern. FormFiller has an AutoRun (or batch) mode that processes each document in the corpus, looking for matches, and it writes the results to an output file. This mode was used when testing FormFiller.

Both processing time and memory usage was observed for each application.

2.3 Determining Baseline and Fact Extractor Performance

The second test involved running different Fact Extractors over both test corpora in the FormFiller application and measuring the performance. A baseline was produced that provided an indication of FormFiller's initial performance, as well as a reference for comparison with later test results. The baseline was developed using fact extractors with differing numbers of create rules, matches, and levels of subordinate fact extractors.

These tests were run over the '10largedocs' corpus as well as the '1000smalldocs' corpus.

2.3.1 Number of Create Rules

The number of create rules [1] in a Fact Extractor was tested to understand the effects of increasing the number of create rules on both processing time and memory use. The following Fact Extractors were used:

- **Disease1.fx** – 1 create rule
- **Disease80.fx** – 80 create rules
- **Disease160.fx** – 160 create rules (double 80)
- **Disease320.fx** – 320 create rules (double 160, 4 times 80)

2.3.2 Number of Matches

A comparison was made of the processing time required to run a Fact Extractor known to produce a high number of matches in the PROMED corpus against that of a Fact Extractor known to find only a few matches in the corpus.

The results of this test were used to analyse FormFiller performance.

The following Fact Extractors were used:

- **Death.fx** – low occurrences (270 matches in **1000smalldocs** corpus)
- **The.fx** – high occurrences (14,643 matches in **1000smalldocs** corpus)

2.3.3 Number of Levels of Subordinate Fact Extractors

A test was done to determine if calling a subordinate Fact Extractor [1] added any significant increase in processing time. The following Fact Extractors were tested:

- **Diseasesub1.fx** – This calls the 'Disease1.fx' Fact Extractor
- **Diseasesub2.fx** – Calls the 'Diseasesub1.fx' Fact Extractor
- **Diseasesub3.fx** – Calls the 'Diseasesub2.fx' Fact Extractor

The results of these tests were also compared with results of the 'Disease1.fx' Fact Extractor[3], as this was the lowest level Fact Extractor.

2.3.4 List Fact Extractors

This test was aimed at comparing processing times for differing formats of the same Fact Extractor.

The *Disease320.fx* Fact Extractor was used for this test. It was also rewritten as a list fact extractor [1], and as a fact extractor with all of the diseases in a single create rule in an "OR" statement. The results of these three types of Fact Extractors were compared to see which format was the fastest performer.

- **Disease320.fx** – 320 individual create rules
- **Disease320in1rule.fx** – 1 create rule with 320 diseases in an OR statement
- **Diseaselist.fx** – A list of 320 diseases

The number of rules was reduced by half for each test. This test was repeated for 160, 80, 40, 20, 10, and 5 create rules, and their respective results were compared to determine which fact extractor type was better used over a certain number of create rules.

2.3.5 Complex Fact Extractors

This test looked at the effect of using a more complex Fact Extractor on processing times.

The fact extractor, *DEF_UnboundedPerson.fx*, was used for this test as it had both create and co-reference rules.

[3] Refer to Section 2.3.1

2.3.6 Different Output Types

While the initial tests concentrated on saving extracted facts to XML output, FormFiller can also save facts in a Comma Separated Values (CSV) file format, as well as in a database. To avoid any overhead associated with accessing a remote database, a Microsoft Access database was configured on the test machine and used for this test.

2.3.7 HTML Input

The tests described in sections 2.3.1 – 2.3.6 were initially performed on a set of text documents, and repeated for the HTML version of the same document set.

2.4 Test System

Every test was performed on a computer with the following specifications to ensure consistency between the results:

Processor (CPU):	Intel® Pentium® 4 2.40GHz
Memory (RAM):	512 MB
Operating System:	Microsoft® Windows™ XP Professional

In calculating the baseline, each test was performed three times, and an average taken to provide a better indication of results.

2.4.1 Testing Processing Time

Processing time was the primary focus of all of the tests. The system time was taken right before and after the required processing and the difference calculated. The TesterGUI application has this function built-in, and the FormFiller application was slightly modified to take the time immediately before and after the AutoRun method is called. At the end of the processing, both the number of documents processed and the total time taken was displayed.

As the time was taken immediately before and after processing, the time to set up and configure the application was not included.

Time was tested independently from memory so that the Task Manager used to monitor the memory usage did not compromise results. No other applications were running apart from the one being tested.

2.4.2 Testing Memory Usage

Although memory usage had not been a problem in the FormFiller application, it was also recorded during baseline measurements in addition to processing time. This was in case later changes to the source code significantly altered memory behaviour.

The Microsoft Windows XP operating system allows users to monitor the memory used by each process by invoking the Windows Task Manager and viewing the 'Processes' tab. The memory used by each process displayed in Task Manager is updated once a second, which means that the results are approximate.

Memory was tested from immediately before the AutoRun process was initiated (start) until it completed (end) as to exclude then memory usage when setting up and configuring the FormFiller application. Again, no other applications were running during this test apart from the one being tested and the task manager.

3. Results of Estimation of Theoretical Lower Bound

These are the averages[4] of the measurements taken when using TesterGUI to determine the performance of the Java REGEX pattern matcher, in order to determine what the best possible performance can be.

Table 1. Results of Test 1 – Java REGEX vs. FormFiller (Averages)

	Time (minutes)	**Memory (MB) at start**	**Memory (MB) at end**
Java REGEX	1.89	11.9	18.7
FormFiller	7.43	25.7	39

The fact that TesterGUI, the program designed to test the Java REGEX pattern matcher, performed so slowly was unexpected, so another REGEX package, *grep*, was looked at.

A free 30-day trial version of grep called PowerGREP [3] was downloaded from http://www.powergrep.com and the *datepattern* regular expression was compared against the '10largedocs' corpus. While this program had no timing function, it was very fast, taking less than half a second to find and report matches.

Inspection of the source code for TesterGUI showed that it had a lot of processing that was not required when attempting to measure the performance of the pattern matcher. A program designed specifically for testing only the Java pattern matcher and nothing else, called **regexTest**, was developed and run with the same regular expression and corpus, and this produced a time of 0.441 seconds, similar to that of PowerGREP.

[4] Refer to Appendix for full results of these tests

Table 2. Comparison between pattern matchers, FormFiller and TesterGUI

	Time
FormFiller	446.074 seconds
TesterGUI	113.634 seconds
PowerGREP	≈ 0.5 seconds
RegexTest (basic Java pattern matcher)	0.441 seconds

From these results, it was clear that the Java REGEX package is not the reason why FormFiller and TesterGUI performed slowly. The magnitude (approximately 1000:1) of the performance difference between FormFiller and REGEX packages was surprising and clearly indicated that there was scope for performance tuning.

4. Initial Baseline Results

Both time and memory usage were recorded during the baseline testing.

The baseline tests were done on fact extractors with different numbers of: create rules, matches, and levels of subordinate fact extractors. The reason that these were the only tests completed initially is because it was known that FormFiller needed improvement, and these tests were done to produce a baseline that any improvements could then be verified against. All of these tests were run in FormFiller, in the AutoRun mode.

As the results show, each test took several minutes, which was very slow. It is also notable that it took longer to process the 10 larger documents compared to the 1000 smaller ones despite the 10 large documents corpus being smaller in total size[5].

These are the averages[6] recorded during these tests:

4.1.1 Number of Create Rules

Tables 3 and 4 show the average processing times and memory usage for Fact Extractors with different numbers of create rules. From these results it can be seen that having a larger number of create rules does not have a significant impact on either processing time or memory usage.

[5] **10largedocs** was 869 KB in total size, compared to 2.39 MB for **1000smalldocs**
[6] Refer to Appendix for full results of all tests

Table 3. Average Times for the Baseline test with varying number of Create Rules

No. of rules	Time (min) over 1000smalldocs	Time (min) over 10largedocs
1	4.02	7.3
80	4.49	7.54
160	5.03	7.43
320	5.94	7.74

Table 4. Average Memory usage for the Baseline test with varying number of Create Rules

No. of rules	Memory (MB) 1000smalldocs	Memory (MB) 10largedocs
1	25 (start) - 36.7 (end)	25.8 (start) - 39.8 (end)
80	25.6 - 33.7	25.6 - 39.4
160	26.2 - 32.3	26.3 - 41
320	26.3 - 35.2	26.9 - 40.4

4.1.2 Number of Matches

Tables 5 and 6 show that when many facts are discovered there is a notable increase in processing times with the small documents. The impact on memory usage was expected.

Table 5. Average Times for Baseline test with varying number of Matches

No. of Matches	Time (min) over 1000smalldocs	Time (min) over 10largedocs
Low	4.04	7.52
High	10.08	7.48

Table 6. Average Memory usage for Baseline test with varying number of Matches

No. of Matches	Memory (MB) of 1000small	Memory (MB) of 10Largedocs
Low	25.9 (start) - 37.9 (end)	26.1 (start) – 41 (end)
High	26.2 - 53.4	25.9 - 40.6

4.1.3 Subordinate Fact Extractors

Tables 7 and 8 show that the different levels of subordinate Fact Extractors have no significant impact either on processing times or memory usage.

Table 7. Average Times for Baseline test with varying levels of Subordinate Fact Extractors

No. Subordinate	Time (min) over 1000smalldocs	Time (min) over 10largedocs
0 (original)	4.02	7.3
1	4.1	7.34
2	4.12	7.32
3	4.1	7.2

Table 8. Average Memory usage for Baseline test with varying levels of Subordinate Fact Extractors

No. Subordinate	Memory (MB) 1000smldocs	Memory (MB) 10Lgdocs
0 (original)	25 (start) - 36.7 (end)	25.8 (start) - 39.8 (end)
1	25.9 - 37.8	26.1 - 40.8
2	26.2 - 37.6	26.2 - 40.5
3	25.7 - 37.9	26 - 40.5

5. Performance Analysis and Tuning

5.1 Analysis Tool and Techniques

The aim of the performance tuning was to reduce the processing times taken. A goal was set for FormFiller to be able to process 1000 small documents within a minute, as this was decided to be an acceptable level of performance.

A Java execution profiler was used to find out where the application was spending most of the time. This is a special program that examines your program while it is running and reports on the time spent in each part of it.

A Java profiler called Borland OptimizeIt [4] (version 4.11) was used for the analysis. It has a CPU profiler that is effective in reporting the time spent in each method, and also displaying it as a percentage of the overall time.

Once a particular time intensive method or piece of code had been identified, the source code was then examined and alternative ways to write it were investigated.

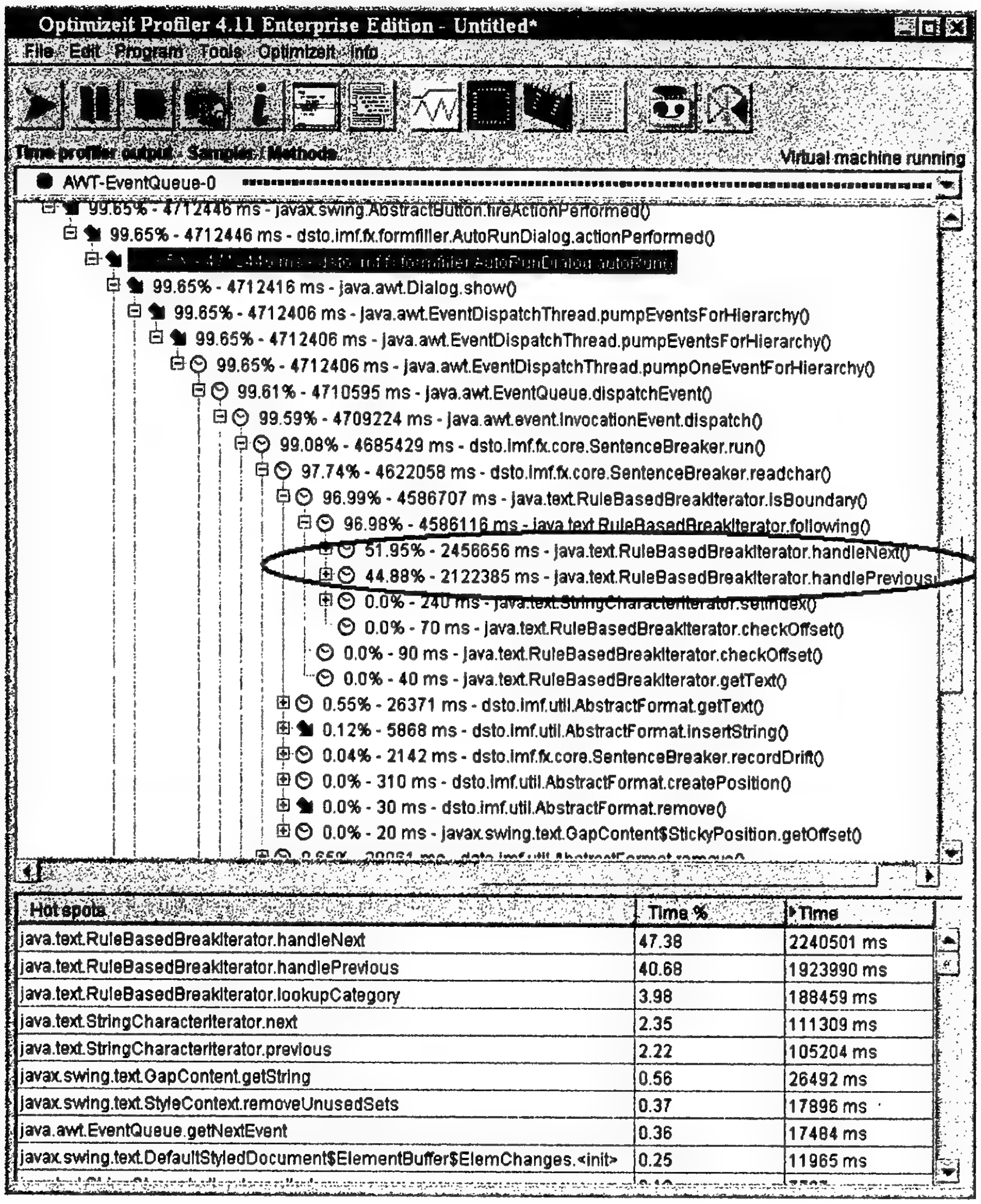
5.2 Initial Performance Optimisation

After using OptimizeIt to monitor 'testdate.fx' being run over '10largedocs' in FormFiller using AutoRun, it was seen that the program spent the majority of its time in

```
dsto.imf.fx.core.SentenceBreaker.run()
```

Figure 1 shows the output of the CPU profiler in OptimizeIt, displaying the problem methods:

Figure 1 – OptimizeIt Screen


Optimizeit Profiler 4.11 Enterprise Edition - Untitled*
File Edit Program Tools OptimizeIt Info
Time profiler output - Samples / Methods
Virtual machine running
AWT-EventQueue-0

```
99.65% - 4712446 ms - javax.swing.AbstractButton.fireActionPerformed()
  99.65% - 4712446 ms - dsto.imf.fx.formfiller.AutoRunDialog.actionPerformed()
    99.65% - 4712446 ms - dsto.imf.fx.formfiller.AutoRunDialog.autoRun()
      99.65% - 4712416 ms - java.awt.Dialog.show()
        99.65% - 4712406 ms - java.awt.EventDispatchThread.pumpEventsForHierarchy()
          99.65% - 4712406 ms - java.awt.EventDispatchThread.pumpEventsForHierarchy()
            99.65% - 4712406 ms - java.awt.EventDispatchThread.pumpOneEventForHierarchy()
              99.61% - 4710595 ms - java.awt.EventQueue.dispatchEvent()
                99.59% - 4709224 ms - java.awt.event.InvocationEvent.dispatch()
                  99.08% - 4685429 ms - dsto.imf.fx.core.SentenceBreaker.run()
                    97.74% - 4622058 ms - dsto.imf.fx.core.SentenceBreaker.readchar()
                      96.99% - 4586707 ms - java.text.RuleBasedBreakIterator.isBoundary()
                        96.98% - 4586116 ms - java.text.RuleBasedBreakIterator.following()
                          51.95% - 2456656 ms - java.text.RuleBasedBreakIterator.handleNext()
                          44.88% - 2122385 ms - java.text.RuleBasedBreakIterator.handlePrevious
                          0.0% - 240 ms - java.text.StringCharacterIterator.setIndex()
                          0.0% - 70 ms - java.text.RuleBasedBreakIterator.checkOffset()
                        0.0% - 90 ms - java.text.RuleBasedBreakIterator.checkOffset()
                        0.0% - 40 ms - java.text.RuleBasedBreakIterator.getText()
                      0.55% - 26371 ms - dsto.imf.util.AbstractFormat.getText()
                      0.12% - 5868 ms - dsto.imf.util.AbstractFormat.insertString()
                      0.04% - 2142 ms - dsto.imf.fx.core.SentenceBreaker.recordDrift()
                      0.0% - 310 ms - dsto.imf.util.AbstractFormat.createPosition()
                      0.0% - 30 ms - dsto.imf.util.AbstractFormat.remove()
                      0.0% - 20 ms - javax.swing.text.GapContent$StickyPosition.getOffset()
```

Hot spots	Time %	Time
java.text.RuleBasedBreakIterator.handleNext	47.38	2240501 ms
java.text.RuleBasedBreakIterator.handlePrevious	40.68	1923990 ms
java.text.RuleBasedBreakIterator.lookupCategory	3.98	188459 ms
java.text.StringCharacterIterator.next	2.35	111309 ms
java.text.StringCharacterIterator.previous	2.22	105204 ms
javax.swing.text.GapContent.getString	0.56	26492 ms
javax.swing.text.StyleContext.removeUnusedSets	0.37	17896 ms
java.awt.EventQueue.getNextEvent	0.36	17484 ms
javax.swing.text.DefaultStyledDocument$ElementBuffer$ElemChanges.<init>	0.25	11965 ms

Further examination showed that this method called the `SentenceBreaker.readChar()` method, which called `java.text.RuleBasedBreakIterator.isBoundary()`. This had two methods of its own: `handleNext()` and `handlePrevious()`, both of which were very time-intensive.

An examination of the source code of '`dsto.imf.fx.core.SentenceManager`' showed that `isBoundary()` was being called to check if every single character was a sentence boundary for each document for each fact extractor.

Since `isBoundary()` was a time-intensive process, instead of repeating it many times, an idea for performance enhancement was to perform it only once over each document, storing the results in a cache. Further on during processing, each character position would be tested to see if it is in the cache, i.e. if it is in the cache it is a sentence break.

A cache was made using a `HashSet` in Java, which is an implementation of the `Set` interface. In theory, this should be an improvement, because this cache uses hashing, which is known to be a fast process. Once the code was modified, the same timing tests were performed again. The results are shown in Tables 9 – 11.

5.3 Results from Initial Optimisation

Table 9. Processing times after initial optimisation for Number of Create Rules

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	29	8.8
80	49	11
160	71.9	13.1
320	117.9	19.2

Table 10. Processing times after initial optimisation for Number of Matches

No. of Matches	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
Low	26.4	8.8
High	388.2 = 6.47 minutes	10.1

Table 11. Processing times after initial optimisation for number of levels of Subordinate Fact Extractors

No. Subordinate	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
0 (original)	29	8.8
1	29.9	8.7
2	27.5	8.6
3	28	9

As is evident from comparing these results with the unoptimised times given in Tables 3, 5 and 7, this change had a dramatic improvement on performance. The time was reduced from minutes to seconds. In the case of a single create rule over 1000 small documents, the time was reduced from 4.02 minutes to 29 seconds, which is 8 times faster. Over 10 large documents, the time improved from 7.3 minutes to 8.8 seconds, which is 50 times faster.

After the initial optimisation, a memory test was run again that showed that the optimisation had no significant effect on the memory usage.

This improvement to the code explains why TesterGUI also took so long, because it implemented the `SentenceBreaker` class in the same way as FormFiller.

The goal of processing 1000 documents within a minute was not fully achieved. In particular, running the fact extractor with a high number of matches ('The.fx') took over 6 minutes. This was run through OptimizeIt to identify any further bottlenecks.

5.4 Second Performance Optimisation

After running 'The.fx' over '1000smalldocs' in OptimizeIt, it was discovered that the majority of the time, 81.72%, was spent in

```
dsto.imf.fx.formfiller.AutoRunDialog.writeXML
```

After that 'Disease320.fx' was run over '1000smalldocs' in OptimizeIt, and again `autoRunDialog.writeXML` took the highest percentage of time, however this time it was only 31.1%. This was most likely because there were fewer facts that needed to be output in the latter case.

Examination of the source code indicated that each time facts were written to the XML output file, the program would check to see if the file existed, then the entire document was read in and the last line checked to see if there was anything there, or if it needed to first write an XML header. So as more and more facts were added, the document became larger, taking longer to read each successive time.

Both the XML and CSV methods were modified to use a `BufferedWriter` instead of just a `FileWriter`.

The `writeXML` method was split into 3 separate methods to stop it from doing unnecessary work. The first method was to output an XML heading, the second to output all of the results, and the third to close the final `<results>` tag. This approach did away with the need to read the output file each time facts were written to it, because the header was written at the start. Facts were appended to the document, and when this was completed the results tag was closed.

5.5 Results from Second Optimisation

The same timing tests were again completed once the code had been changed. Tables 12 – 14 show the results from these tests.

Table 12. Times after second optimisation – Number of Create Rules

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	24.4	8.7
80	31.3	10.2
160	38.3	12.7
320	57.6	18.3

Table 13. Times after second optimisation– Number of Matches

No. of Matches	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
Low	23.4	8
High	25.1	8.6

Table 14. Times after second optimisation – Number of levels of Subordinate Fact Extractors

No. subordinate	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
0 (original)	24.4	8.7
1	25.5	8.1
2	27.4	8.1
3	28.2	8.1

As the results indicate, this improvement made a notable difference to processing times, particularly for the Fact Extractors with a large number of create rules and a large number of matches. The second optimisation allowed us to achieve the goal of processing all of the observed times in under a minute.

A further test run in OptimizeIt showed that most of the time was spent in `java.awt.EventQueue.getNextEvent,` which is an idle process. As the goal of processing 1000 small documents within a minute had been achieved, no further optimisation was undertaken. It was decided however that if any futher tests that followed failed the 1000 documents a minute requirement, they would be run through the profiler.

6. Further Testing

Since the initial performance goal had been achieved, a wider set of Fact Extractors was tested in FormFiller (again using AutoRun) to observe their performance. Tests were also run using both text and HTML documents as input files, and with facts being saved to CSV and XML files, as well as to a database.

6.1 Different Types of Fact Extractor

For Fact Extractors with different numbers of create rules, matches, and levels of subordinate Fact Extractors, refer to section 5.5.

Table 15 shows that the processing time for a Fact Extractor with a co-reference rule, 'DEF_UnboundedPerson.fx', is well under the 1-minute upper limit.

Table 15. Time results for a Fact Extractor with a co-reference rule

Time (sec) over 1000 small documents	Time (sec) over 10 large documents
34.8	17.5

6.2 Output Targets

Up to this point, all the tests have been output to XML format. The baseline[7] tests were repeated for both CSV and database output. The results are shown in Tables 16 – 18.

Table 16. Number of Create Rules output to XML

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	24.4	8.7
80	31.3	10.2
160	38.3	12.7
320	57.6	18.3

Table 17. Number of Create Rules output to CSV

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	28.3	8.5
80	30.3	10.6
160	43	13
320	56	19.1

[7] The times for differing number of create rules are shown here only. Refer to Appendix for full results.

Table 18. Number of Create Rules output to MSAccess Database

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	23.7	8.4
80	35.4	12.4
160	45.7	15.3
320	66.2	21.6

These results indicate that there is no significant difference in using either XML or CSV files as the output target. Writing to a database took a little longer for the Fact Extractors with a large number of create rules. Although it took 66.2 seconds to write the facts found with the 320-create rule Fact Extractor to the database, this was accepted because it was a Microsoft Access database on the test machine. The extra time was spent inserting the facts into the database.

6.3 List Fact Extractors

For all of the previous tests, each disease found in the Fact Extractors had its own individual create rule. This is one format of a list fact extractor, and on the graphs below is represented as 'Individual Rules'. The list of diseases was also made into a List Fact Extractor, which was a text file that had each disease on its own line. On the following graphs, this is shown as 'List'. The third format of Fact Extractor had all of the diseases combined into an 'OR' statement, and this statement made up a single create rule. This is shown on the graphs below as 'One Big Rule'.

After completing the tests, the following comparison graphs were produced[8].

[8] Refer to Appendix for the full results

Figure 2 – Different FX formats comparison over 1000 small documents

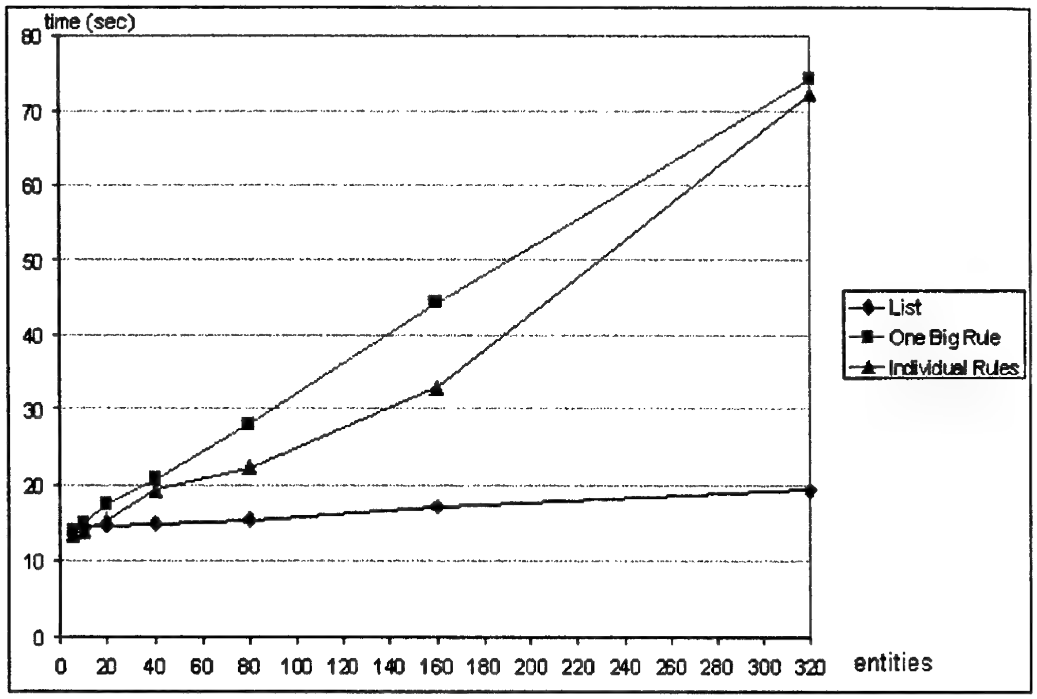
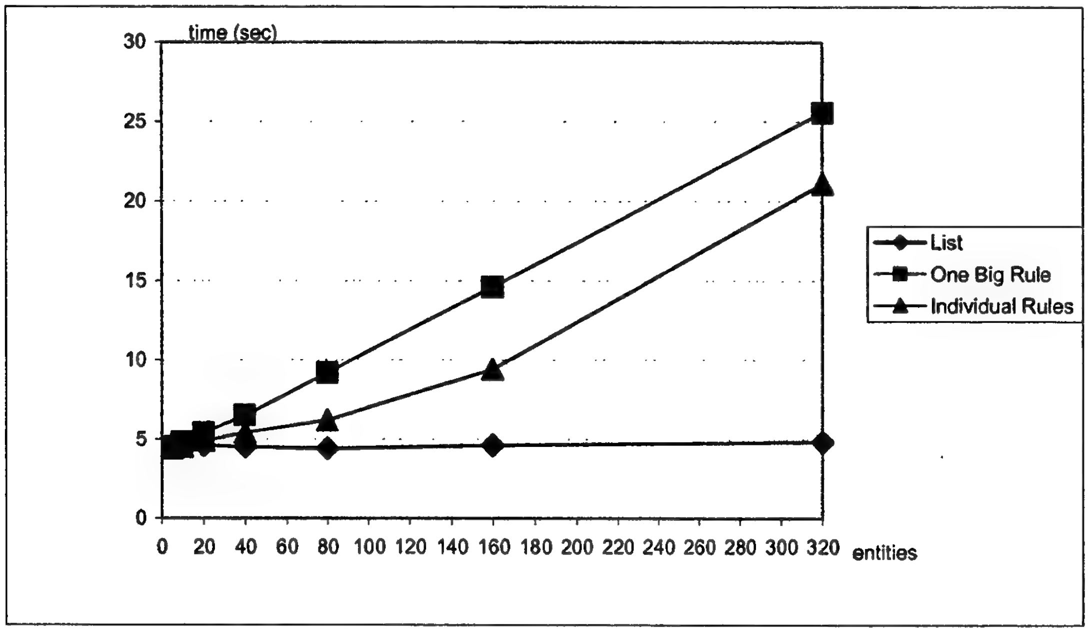


Figure 3 – Different FX formats comparison over 10 large documents



Figures 2 and 3 show that as the number of rules increased, the List Fact Extractor had the most efficient processing times over both sets of documents.

6.4 Input Sources

So far, all the tests had used text file input. This was then compared to HTML input. The HTML input files were made up of exactly the same data as the text input, only in HTML format. Tables 19 and 20[9] show that processing HTML input takes slightly longer than text input, particularly for the larger documents.

Table 19. Times for text input with differing number of Create Rules

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	24.4	8.7
80	31.3	10.2
160	38.3	12.7
320	57.6	18.3

Table 20. Times for HTML input with differing number of Create Rules

No. of rules	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
1	30.6	28.4
80	36.9	29.8
160	42.5	32
320	57	37.2

7. Conclusion

The performance analysis and optimisation of the Fact Extractor System has proved to be highly beneficial allowing the goal time of processing 1000 documents within a minute to be achieved. The use of the OptimizeIt profiler was useful in quickly identifying problem areas. The improvements that were made to the sentence breaking in particular will benefit the entire suite of Fact Extraction tools, because this was one of the core components.

As no further major bottlenecks were identified in the final set of tests the results of those tests can now be used to help design effective fact extractors.

The test results clearly showed that adding more create rules to a fact extractor does not significantly increase the processing time. As the number of rules was doubled, the

[9] Refer to Appendix for full results

time did not double. Processing 320 rules over 1000 documents still takes less than one minute, even in the unlikely event that a Fact Extractor contained that many rules.

The results also show that the actual number of facts found is of little significance to the processing time. It does not take much extra time to process a Fact Extractor that finds a very high number of facts as opposed to one that finds a much lower number. The majority of processing time is spent evaluating the rules.

Calling subordinate Fact Extractors similarly has little impact on the processing time. Therefore, for ease of development, if a user invokes a subordinate Fact Extractor instead of making a complex rule, the overhead in terms of extra processing time will be minimal and will not depend on how many levels of subordinate Fact Extractors are invoked.

The more complex Fact Extractor used to test the presence of co-reference rules had two create rules as well as the co-reference rule. The time taken to process this was the equivalent of processing over 80 create rules. This indicated a processing time overhead for the co-reference rule, however the time was still within the 1000 documents per minute target.

List fact extractors are a lot faster to process than ones based on multiple create rules. If there are any more than about 10 rules to search for, it would be most efficient to put them in a list. However, the list processor does not resolve regular expressions, it will only search for exact matches of the text included in the list, and so this type of fact extractor is best suited for entities, such as place or organisation names.

There is no major difference to processing times for saving extracted facts to either XML or CSV files, however writing to a database takes a little longer.

HTML input takes a little longer to process than text input, and this is because HTML is a more complex document format, however it does not have a significant impact on processing time.

In addition to the original 'PROMED' documents, a different corpus of 1000 documents was tested to make sure that FormFiller could process these within the target time of less than one minute. The performance target was met with the new corpus.

For the test corpora and test set of Fact Extractors, memory usage was never an issue both before and after the optimisation.

Experience with the testing suggested some improvements for the test process.

- The test corpora should have been the same total size. Instead, '10largedocs' was 869 KB and '1000smalldocs' was 2.39 MB.

- More complex Fact Extractors should have been tested. Testing more complex Fact Extractors would have given a more rigorous set of timing results, and may have revealed additional problem areas in the software.

8. References

[1] J. Das, G. Chase and S. Davis, "Fact Extractor System Processing Engine", DSTO-TR-1396.

[2] https://sourceforge.dsto.defence.gov.au/projects/factextractors/ , last accessed 25 June 2004

[3] http://www.powergrep.com , last accessed 25 June 2004

[4] http://www.borland.com/optimizeit/ , last accessed 25 June 2004

Appendix: Full Test Results

A.1 Before Optimisation

A.1.1 Test One – Java REGEX Pattern Matcher vs. FormFiller

A.1.1.1 Time

Table 21. Full time results for Test 1 – Java REGEX vs. FormFiller

Test	FormFiller Time	Java REGEX Time
1	447 584 ms = 7.46 minutes	121 004 ms = 2.0167 minutes
2	443 748 ms = 7.40 minutes	110 009 ms = 1.8335 minutes
3	446 892 ms = 7.45 minutes	109888 ms = 1.8315 minutes
Average	446 074 ms = 7.43 minutes	113634 ms = 1.8939 minutes

A.1.1.2 Memory

Table 22. Full memory results for Test 1 – Java REGEX vs. FormFiller

Test	FormFiller Memory (start – end)	Java REGEX Memory (start – end)
1	26 160 KB – 39 356 KB	12 224 KB – 19 116 KB
2	26 080 KB – 40 712 KB	12 224 KB – 19 364 KB
3	26 712 KB – 40 044 KB	12 228 KB – 19 112 KB
Average	26 317 KB – 40 037 KB	12 225 KB – 19 197 KB

A.1.2 Test Two – Some Comparisons of Different Fact Extractors

A.1.2.1 Time

A.1.2.1.1 Number of Create Rules

Table 23. Full time results for 1 Create Rule

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	245984 = 4.1 minutes	436498 = 7.3 minutes
2	239855 = 4 minutes	445601 = 7.4 minutes
3	238082 = 4 minutes	432713 = 7.2 minutes
Average	241307 = 4.02 minutes	438271 = 7.3 minutes

Table 24. Full time results for 80 Create Rules

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	274274 = 4.6 minutes	460593 = 7.7 minutes
2	269868 = 4.5 minutes	445441 = 7.4 minutes
3	264911 = 4.4 minutes	450327 = 7.5 minutes
Average	269684 = 4.5 minutes	452120 = 7.54 minutes

Table 25. Full time results for 160 Create Rules

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	301083 = 5 minutes	446142 = 7.43 minutes
2	292240 = 4.9 minutes	446322 = 7.44 minutes
3	312700 = 5.2 minutes	444479 = 7.4 minutes
Average	302008 = 5.03 minutes	445648 = 7.43 minutes

Table 26. Full time results for 320 Create Rules

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	355421 = 5.92 minutes	472299 = 7.9 minutes
2	353729 = 5.9 minutes	448795 = 7.5 minutes
3	359838 = 6 minutes	472389 = 7.9 minutes
Average	356329 = 5.94 minutes	464494 = 7.74 minutes

A.1.2.1.2 Number of Matches

The high number of matches finds 14, 643 facts in the 1000 documents corpus, and 5819 in the 10 documents corpus.

The low number of matches finds 270 facts in the 1000 documents corpus, and 139 in the 10 documents corpus.

Table 27. Full time results for high number of matches

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	618510 = 10.3 minutes	441024 = 7.4 minutes
2	604409 = 10.1 minutes	466551 = 7.8 minutes
3	590649 = 9.8 minutes	439171 = 7.3 minutes
Average	604523 = 10 minutes	448949 = 7.48 minutes

Table 28. Full time results for low number of matches

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	243560 = 4.1 minutes	453622 = 7.6 minutes
2	242239 = 4 minutes	442536 = 7.4 minutes
3	242238 = 4 minutes	457027 = 7.6 minutes
Average	242679 = 4.04 minutes	451062 = 7.52 minutes

A.1.2.1.3 Subordinate Fact Extractors

Table 29. Full time results for the bottom level fact extractor

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	245984 = 4.1 minutes	436498 = 7.3 minutes
2	239855 = 4 minutes	445601 = 7.4 minutes
3	238082 = 4 minutes	432713 = 7.2 minutes
Average	241307 = 4.02 minutes	438271 = 7.3 minutes

Table 30. Full time results for 1 subordinate fact extractor

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	249569 = 4.2 minutes	444169 = 7.4 minutes
2	244171 = 4 minutes	439922 = 7.3 minutes
3	244041 = 4 minutes	438080 = 7.3 minutes
Average	245927 = 4.1 minutes	440724 = 7.35 minutes

Table 31. Full time results for 2 subordinate fact extractors

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	248006 = 4.1 minutes	443638 = 7.4 minutes
2	242068 = 4 minutes	434265 = 7.2 minutes
3	252182 = 4.2 minutes	439623 = 7.3 minutes
Average	247419 = 4.12 minutes	439175 = 7.32 minutes

Table 32. Full time results for 3 subordinate fact extractors

Test	Time (ms) 1000smalldocs	Time (ms) 10largedocs
1	245724 = 4.1 minutes	439201 = 7.3 minutes
2	244602 = 4.1 minutes	434445 = 7.2 minutes
3	246474 = 4.1 minutes	431851 = 7.2 minutes
Average	245600 = 4.09 minutes	435166 = 7.25 minutes

A.1.2.2 Memory

A.1.2.2.1 Number of Create Rules

Table 33. Memory results for 1 create rule

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	25488 (start) – 37612 (end)	26064 (start) – 40868 (end)
2	25848 – 37592	26600 – 41148
3	25464 – 37608	26624 – 40276
Average	25600 – 37604	26429 – 40764

Table 34. Memory results for 80 create rules

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26220 (start) – 34204 (end)	26044 (start) – 39352 (end)
2	25880 – 34680	26632 - 41100
3	26396 – 34752	25844 – 40612
Average	26165 - 34545	26173 – 40355

Table 35. Memory results for 160 create rules

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26888 (start) – 33516 (end)	26540 (start) – 42080 (end)
2	26708 – 32704	27384 – 42228
3	26920 – 32972	26756 – 41660
Average	26839 – 33064	26893 – 41989

Table 36. Memory results for 320 create rules

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26628 (start) – 36316 (end)	27464 (start) – 41784 (end)
2	26936 – 35968	27440 – 40548
3	27088 – 35868	27744 – 41792
Average	26884 – 36051	27549 – 41375

A.1.2.2.2 Number of Matches

Table 37. Memory results for high number of matches

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26788 (start) – 54572 (end)	26608 (start) – 41192 (end)
2	26800 – 54788	26880 – 42504
3	25832 – 54788	26544 – 41004
Average	26473 – 54716	26677 – 41576

Table 38. Memory results for low number of matches

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26312 (start) – 40680 (end)	26728 (start) – 40752 (end)
2	28080 – 36372	26396 – 41660
3	26352 – 39340	26532 – 43572
Average	26855 – 38797	26552 – 41995

A.1.2.2.3 Subordinate Fact Extractors

Table 39. Memory results for bottom-level fact extractor

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	25488 (start) – 37612 (end)	26064 (start) – 40868 (end)
2	25848 – 37592	26600 – 41148
3	25464 – 37608	26624 – 40276
Average	25600 – 37604	26429 – 40764

Table 40. Memory results for 1 subordinate fact extractor

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26160 (start) – 38584 (end)	26832 (start) – 41212 (end)
2	26720 – 39012	26452 – 41356
3	26640 – 38636	26828 – 42620
Average	26507 – 38744	26701 – 41729

Table 41. Memory results for 2 subordinate fact extractors

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	27396 (start) – 38744 (end)	26768 (start) – 42268 (end)
2	25892 – 38696	26768 – 41864
3	27172 – 38192	26880 – 40188
Average	26820 – 38544	26805 – 41440

Table 42. Memory results for 3 subordinate fact extractors

Test	Memory (KB) 1000smalldocs	Memory (KB) 10largedocs
1	26180 (start) – 38088 (end)	26852 (start) – 42164 (end)
2	26576 – 37392	26484 – 41700
3	26292 – 38800	26440 – 41468
Average	26349 – 38093	26592 – 41777

A.2 After First Optimisation

Table 43. Time Results of Number of Create Rules

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	29082 = 29 seconds	8873 = 8.8 seconds
80	49261 = 49 seconds	10965 = 11 seconds
160	71874 = 71.9 seconds	13079 = 13.1 seconds
320	117990 = 117.9 seconds	19238 = 19.2 seconds

Table 44. Time results for Number of Matches

No. of Matches	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
Low	26377 = 26.4 seconds	8883 = 8.8 seconds
High	388389 = 6.47 minutes	10134 = 10.1 seconds

Table 45. Time Results of Subordinate Fact Extractors

No. subordinate	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
0 (original)	29082 = 29 seconds	8873 = 8.8 seconds
1	29923 = 29.9 seconds	8713 = 8.7 seconds
2	27470 = 27.5 seconds	8652 = 8.7 seconds
3	28381 = 28 seconds	9023 = 9 seconds

A.3 After Second Optimisation

Table 46. Time Results of Number of Create Rules

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	24485 = 24.4 seconds	8697 = 8.7 seconds
80	31315 = 31.3 seconds	10154 = 10.2 seconds
160	38305 = 38.3 seconds	12678 = 12.7 seconds
320	57603 = 57.6 seconds	18346 = 18.3 seconds

Table 47. Time results for Number of Matches

No. of Matches	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
Low	23423 = 23.4 seconds	8002 = 8 seconds
High	25107 = 25.1 seconds	8593 = 8.6 seconds

Table 48. Time Results of Subordinate Fact Extractors

No. subordinate	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
0 (original)	24485 = 24.4 seconds	8697 = 8.7 seconds
1	25526 = 25.5 seconds	8072 = 8.1 seconds
2	27449 = 27.4 seconds	8062 = 8.1 seconds
3	28200 = 28.2 seconds	8111 = 8.1 seconds

A.4 Further Tests

A.4.1 Different Type of Fact Extractor

Create rules and a co-reference rule.

Table 49. Time Results for Number of Create Rules

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	24485 = 24.4 seconds	8697 = 8.7 seconds
80	31315 = 31.3 seconds	10154 = 10.2 seconds
160	38305 = 38.3 seconds	12678 = 12.7 seconds
320	57603 = 57.6 seconds	18346 = 18.3 seconds

Table 50. Time results for Number of Matches

No. of Matches	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
Low	23423 = 23.4 seconds	8002 = 8 seconds
High	25107 = 25.1 seconds	8593 = 8.6 seconds

Table 51. Time Results for Levels of Subordinate Fact Extractors

No. subordinate	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
0 (original)	24485 = 24.4 seconds	8697 = 8.7 seconds
1	25526 = 25.5 seconds	8072 = 8.1 seconds
2	27449 = 27.4 seconds	8062 = 8.1 seconds
3	28200 = 28.2 seconds	8111 = 8.1 seconds

Table 52. Time results for a Fact Extractor with a co-reference rule

Time (ms) over 1000 small documents	Time (ms) over 10 large documents
34780 = 34.8 seconds	17475 = 17.5 seconds

A.4.2 Different Output Types

Table 53. Number of Create Rules output to XML

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	24485 = 24.4 seconds	8697 = 8.7 seconds
80	31315 = 31.3 seconds	10154 = 10.2 seconds
160	38305 = 38.3 seconds	12678 = 12.7 seconds
320	57603 = 57.6 seconds	18346 = 18.3 seconds

Table 54. Number of Matches output to XML

No. of Matches	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
Low	23423 = 23.4 seconds	8002 = 8 seconds
High	25107 = 25.1 seconds	8593 = 8.6 seconds

Table 55. Levels of Subordinate Fact Extractors output to XML

No. subordinate	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
0 (original)	24485 = 24.4 seconds	8697 = 8.7 seconds
1	25526 = 25.5 seconds	8072 = 8.1 seconds
2	27449 = 27.4 seconds	8062 = 8.1 seconds
3	28200 = 28.2 seconds	8111 = 8.1 seconds

Table 56. Number of Create Rules output to CSV

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	28271 = 28.3 seconds	8482 = 8.5 seconds
80	30304 = 30.3 seconds	10555 = 10.5 seconds
160	43072 = 43.1 seconds	13029 = 13 seconds
320	56421 = 56.4 seconds	19158 = 19.2 seconds

Table 57. Number of Matches output to CSV

No. of matches	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
Low	16083 = 16.1 seconds	4296 = 4.3 seconds
High	19468 = 19.5 seconds	5368 = 5.4 seconds

Table 58. Subordinate Fact Extractors output to CSV

No. subordinate	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
0 (original)	28271 = 28.3 seconds	8482 = 8.5 seconds
1	26539 = 26.5 seconds	8833 = 8.8 seconds
2	27189 = 27.2 seconds	8953 = 9 seconds
3	25267 = 25.3 seconds	8432 = 8.4 seconds

Table 59. Number of Create Rules output to Database

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	23714 = 23.7 seconds	8402 = 8.4 seconds
80	35371 = 35.4 seconds	12358 = 12.4 seconds
160	45746 = 45.7 seconds	15322 = 15.3 seconds
320	66185 = 66.2 seconds	21601 = 21.6 seconds

Table 60. Number of Matches output to Database

No. of matches	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
Low	24806 = 24.8 seconds	8372 = 8.4 seconds
High	72664 = 72.7 seconds	29682 = 29.7 seconds

Table 61. Subordinate Fact Extractors output to Database

No. subordinate	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
0 (original)	23714 = 23.7 seconds	8402 = 8.4 seconds
1	26308 = 26.3 seconds	8142 = 8.1 seconds
2	27190 = 27.2 seconds	8121 = 8.1 seconds
3	28010 = 28 seconds	8182 = 8.2 seconds

A.4.3 List Fact Extractors

Table 62. 320 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	19.3	4.8
1 Rule	74.4	25.6
320 Rules	72.3	21.1

Table 63. 160 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	17.2	4.6
1 Rule	44.3	14.6
160 Rules	32.9	9.4

Table 64. 80 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	15.5	4.4
1 Rule	28	9.2
80 Rules	22.3	6.2

Table 65. 40 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	15.1	4.5
1 Rule	20.9	6.5
40 Rules	19.4	5.4

Table 66. 20 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	14.7	4.6
1 Rule	17.5	5.4
20 Rules	15.6	4.9

Table 67. 10 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	14.5	4.6
1 Rule	15.2	4.8
10 Rules	14.0	4.5

Table 68. 5 Rules

FX Format	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
List	13.9	4.4
1 Rule	14.1	4.5
5 Rules	13.6	4.4

A.4.3 Different Input Types

Table 69. Time Results for Number of Create Rules with Text Input

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	24485 = 24.4 seconds	8697 = 8.7 seconds
80	31315 = 31.3 seconds	10154 = 10.2 seconds
160	38305 = 38.3 seconds	12678 = 12.7 seconds
320	57603 = 57.6 seconds	18346 = 18.3 seconds

Table 70. Time results for Number of Matches with Text Input

No. of Matches	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
Low	23423 = 23.4 seconds	8002 = 8 seconds
High	25107 = 25.1 seconds	8593 = 8.6 seconds

Table 71. Time Results for Subordinate Fact Extractors with Text Input

No. subordinate	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
0 (original)	24485 = 24.4 seconds	8697 = 8.7 seconds
1	25526 = 25.5 seconds	8072 = 8.1 seconds
2	27449 = 27.4 seconds	8062 = 8.1 seconds
3	28200 = 28.2 seconds	8111 = 8.1 seconds

Table 72. Time results for a Fact Extractor with a co-reference rule with Text Input

Time (ms) over 1000 small documents	Time (ms) over 10 large documents
34780 = 34.8 seconds	17475 = 17.5 seconds

Table 73. Time Results for Number of Create Rules with HTML Input

No. of rules	Time (ms) over 1000smalldocs	Time (ms) over 10largedocs
1	30594 = 30.6 seconds	28351 = 28.4 seconds
80	36873 = 36.9 seconds	29813 = 29.8 seconds
160	42501 = 42.5 seconds	32016 = 32 seconds
320	56992 = 57 seconds	37243 = 37.2 seconds

Table 74. Time results for Number of Matches with HTML Input

No. of Matches	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
Low	29663 = 29.7 seconds	26519 = 26.5 seconds
High	29993 = 30 seconds	28350 = 28.4 seconds

Table 75. Time Results for Subordinate Fact Extractors with HTML Input

No. subordinate	Time (sec) over 1000smalldocs	Time (sec) over 10largedocs
0 (original)	30594 = 30.6 seconds	28351 = 28.4 seconds
1	31846 = 31.8 seconds	26548 = 26.5 seconds
2	33148 = 33.1 seconds	25570 = 27.6 seconds
3	33138 = 33.1 seconds	27790 = 27.8 seconds

Table 76. Time results for a Fact Extractor with 320 diseases in one OR statement with HTML Input

Time (ms) over 1000 small documents	Time (ms) over 10 large documents
68899 = 68.9 seconds	49802 = 49.8 seconds

Table 77. Time results for a Fact Extractor with a co-reference rule with HTML Input

Time (ms) over 1000 small documents	Time (ms) over 10 large documents
33018 = 33 seconds	35871 = 35.9 seconds

DISTRIBUTION LIST

Performance Analysis and Optimisation of the Fact Extractor System

Shona Heath

(DSTO-TN-0566)

	Number of Copies
AUSTRALIA	
DEFENCE ORGANISATION	
Task sponsor:	
DGICSO, Defence Intelligence Organisation	Doc Data Sheet
DINTCAP, Defence Intelligence Organisation	1
ADIIE, Defence Intelligence Organisation	1
S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development)	
Counsellor, Defence Science, London	Doc Data Sheet
Counsellor, Defence Science, Washington	Doc Data Sheet
Scientific Adviser to MRDC, Thailand	Doc Data Sheet
Scientific Adviser Joint	1
Navy Scientific Adviser	Doc Data Sheet
Scientific Adviser - Army	Doc Data Sheet
Air Force Scientific Adviser	Doc Data Sheet
Scientific Adviser to the DMO M&A	Doc Data Sheet
Scientific Adviser to the DMO ELL	Doc Data Sheet
Information Sciences Laboratory	
Chief Command & Control Division	Doc Data Sheet
Research Leader Command & Intelligence Environments Branch	1
Research Leader Military Information Enterprise Branch	1
Research Leader Theatre Command Analysis Branch	1
Head Intelligence Analysis	1
Head Distributed Enterprises	Doc Data Sheet
Head Systems Simulation and Assessment	1
Head Theatre Operations Analysis	Doc Data Sheet
Head Information Exploitation	Doc Data Sheet
Head Human Systems Integration	Doc Data Sheet

Head C2 Australian Theatre	Doc Data Sheet
Head HQ Systems Experimentation	Doc Data Sheet
Head Information Systems	Doc Data Sheet
S. Heath, C2D (Author)	1
Publications and Publicity Officer, C2D/EOC2D	1 shared copy
J. Das, C2D	1
G. Chase, C2D	1
Dr. C. Rainsford, C2D	1
Dr. T. Pattison, C2D	1
Brendan Dennis, C2D	1
DSTO Library and Archives	
Library Edinburgh	2
Australian Archives	1
Capability Systems Staff	
Director General Maritime Development	Doc Data Sheet
Director General Information Capability Development	Doc Data Sheet
Office of the Chief Information Officer	
Deputy CIO	Doc Data Sheet
Director General Information Policy and Plans	Doc Data Sheet
AS Information Strategies and Futures	Doc Data Sheet
AS Information Architecture and Management	Doc Data Sheet
Director General Australian Defence Simulation Office	Doc Data Sheet
Strategy Group	
Director General Military Strategy	Doc Data Sheet
Director General Preparedness	Doc Data Sheet
Navy	
Director General Navy Capability, Performance and Plans, Navy Headquarters	Doc Data Sheet
Director General Navy Strategic Policy and Futures, Navy Headquarters	Doc Data Sheet
Air Force	
SO (Science) - Headquarters Air Combat Group, RAAF Base, Williamtown NSW 2314	Doc Data Sht & Exec Summ
Army	
ABCA National Standardisation Officer, Land Warfare Development Sector, Puckapunyal	e-mailed Doc Data Sheet
SO (Science), Deployable Joint Force Headquarters (DJFHQ) (L), Enoggera QLD	Doc Data Sheet
SO (Science) - Land Headquarters (LHQ), Victoria Barracks NSW	Doc Data Sht & Exec Summ

Intelligence Program	
COMD ASTJIC, Australian Theatre Joint Intelligence Centre	Doc Data Sheet
PDJISS, Joint Intelligence Support System	Doc Data Sheet
PBD, Defence Signals Directorate	1
PBK, Defence Signals Directorate	Doc Data Sheet
J2 HQAST	Doc Data Sheet
Mr Jeff Robertson, Defence Intelligence Organisation	Doc Data Sheet
SQNLDR Pete Wooding, Capability Systems	Doc Data Sheet
DGSTA Defence Intelligence Organisation	1
Manager, Information Centre, Defence Intelligence Organisation	1 (PDF version)
Assistant Secretary Corporate, Defence Imagery and Geospatial Organisation	Doc Data Sheet
Defence Materiel Organisation	
Head Aerospace Systems Division	Doc Data Sheet
Chief Joint Logistics Command	Doc Data Sheet
Head Materiel Finance	Doc Data Sheet
Defence Libraries	
Library Manager, DLS-Canberra	Doc Data Sheet
Library Manager, DLS-Sydney West	Doc Data Sheet
OTHER ORGANISATIONS	
National Library of Australia	1
NASA (Canberra)	1
UNIVERSITIES AND COLLEGES	
Australian Defence Force Academy	
Library	1
Head of Aerospace and Mechanical Engineering	1
Serials Section (M list), Deakin University Library, Geelong, VIC (Research and Technical Reports only)	1
Hargrave Library, Monash University	Doc Data Sheet
Librarian, Flinders University	1

OUTSIDE AUSTRALIA

INTERNATIONAL DEFENCE INFORMATION CENTRES	
US Defense Technical Information Center	2
UK Defence Research Information Centre	2
Canada Defence Scientific Information Service	1
NZ Defence Information Centre	1
ABSTRACTING AND INFORMATION ORGANISATIONS	
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1
Spare	**5**
Total number of copies:	**43**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE Performance Analysis and Optimisation of the Fact Extractor System	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Shona Heath	5. CORPORATE AUTHOR Information Sciences Laboratory PO Box 1500 Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER DSTO-TN-0566	6b. AR NUMBER AR- 013-124	6c. TYPE OF REPORT Technical Note	7. DOCUMENT DATE June 2004

8. FILE NUMBER E9505/28/40	9. TASK NUMBER INT 02/290	10. TASK SPONSOR DGICSO, DIO DINTCAP, DIO ADIIE, DIO	11. NO. OF PAGES 40	12. NO. OF REFERENCES 4

13. URL on the World Wide Web http://www.dsto.defence.gov.au/corporate/reports/DSTO-TN-0566.pdf	14. RELEASE AUTHORITY Chief, Command and Control Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DEFTEST DESCRIPTORS

Text retrieval
Text processing (computer science)
Performance evaluation

19. ABSTRACT

DSTO has developed an in-house application called the Fact Extractor System for performing Information Extraction. This system can be used to extract interesting information from text documents. It is a component-based system providing a suite of tools to do this task. The system has a number of deployment tools, including one called FormFiller. FormFiller is an application that enables a user to process a set of documents, one at a time interactively or in an automated batch mode, with one or more Fact Extractors and save results to an output file under user control. This report describes performance testing and optimisation of the FormFiller application and Fact Extractors.

Page classification: UNCLASSIFIED